# User's Manual

**Connect to Your Process Control**

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Visual Basic, Microsoft .NET, Visual C++, Visual C#, and ActiveX are either trademarks or registered trademarks of Microsoft Corporation.
PC-LabCard is a trademark of Advantec Co., Ltd.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

Copyright © 2002-2008 by MVTec Software GmbH, München, Germany

| | | |
|---|---|---|
| Edition 1 | November 2002 | (ActivVisionTools 2.1) |
| Edition 2 | January 2005 | (ActivVisionTools 3.0) |
| Edition 3 | February 2006 | (ActivVisionTools 3.1) |
| Edition 4 | May 2008 | (ActivVisionTools 3.2) |

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

More information about ActivVisionTools can be found at:

`http://www.activ-vision-tools.com`

# How to Read This Manual

This manual explains how to use ActivDigitalIO to connect an ActivVisionTools application via digital I/O to a process. It describes the functionality of ActivDigitalIO and its cooperation with other ActivVisionTools with Visual Basic examples. Before reading this manual, we recommend to read the manual Getting Started with ActivVisionTools, which introduces the basic concepts of ActivVisionTools, and the User's Manual for ActivView to learn how to load and display images.

For each example in this manual, there is a corresponding Visual Basic project; these projects can be found in the subdirectory `examples\manuals\activdigitalio` of the ActivVisionTools base directory you selected during the installation (default: `C:\Program Files\MVTec\ActivVisionTools` ). Of course, you can also create your own Visual Basic projects from scratch. Note that the screenshots in the manual may differ slightly from the corresponding Visual Basic projects. To follow the examples actively, first install and configure ActivVisionTools as described in the manual Getting Started with ActivVisionTools.

We recommend to **create a private copy of the example projects** because by experimenting with the projects, you also change their *state*, which is then automatically stored in the so-called description files (extension `.dsc`) by ActivVisionTools. Of course, you can restore the state of a project by retrieving the corresponding description file from the CD.

# Contents


**1 About ActivDigitalIO** **1**

1.1 Introducing ActivDigitalIO . . . . . 2

1.2 The Sub-Tools of ActivDigitalIO . . . . . 4

1.3 A Closer Look at the Interface of ActivVisionTools to Digital I/O . . . . . 6

**2 Using ActivDigitalIO** **13**

2.1 Configuring the Connection to a Digital I/O Board . . . . . 14

2.2 Assigning Activities to Input and Output Channels . . . . . 16

**3 Tips & Tricks** **25**

3.1 Programming Interface . . . . . 26

3.2 How to Connect to Unsupported Digital I/O Boards . . . . . 42

# Chapter 1

# About ActivDigitalIO



This chapter will introduce you to the features and the basic concepts of ActivDigitalIO. It gives an overview about ActivDigitalIO's *master tool* and its *support tools*, which are described in more detail in chapter 2 on page 13.

In section 1.3 on page 6 we take a closer look at the interface that ActivVisionTools provides for digital I/O.

**1.1 Introducing ActivDigitalIO** . . . . . **2**
**1.2 The Sub-Tools of ActivDigitalIO** . . . . . **4**
**1.3 A Closer Look at the Interface of ActivVisionTools to Digital I/O** . . . . . **6**
1.3.1 Components Involved in Digital I/O . . . . . 6
1.3.2 Predefined Digital I/O in ActivVisionTools . . . . . 8
1.3.3 When are Input Channels Checked? . . . . . 11

## 1.1 Introducing ActivDigitalIO

In many machine vision applications, the task is not only to inspect images and calculate certain results but to *output* those results to the surrounding process. The process can then react on them, e.g., remove a faulty part from the conveyor belt. On the other hand, the machine vision application might also need *input* information from the process, e.g., to synchronize its execution to external events.

One way to connect the process to the machine vision application running on a computer is to use a so-called *digital I/O board*, a special hardware (connected, e.g., to the PCI bus of the computer) that provides a certain number of input and output *channels* over which you can send and receive digital signals. The channels are typically optically isolated and can thus be directly connected to the process control.

Using ActivDigitalIO, you can access certain digital I/O boards from Advantech Co., Ltd. directly from within your ActivVisionTools application. Currently, the following boards are supported:

PCI-1710, PCI-1711, PCI-1712, PCI-1716
PCI-1721, PCI-1723
PCI-1730
PCI-1750, PCI-1751, PCI-1753, PCI-1756
PCI-1760, PCI-1762

Besides, ActivDigitalIO provides a so-called *virtual digital I/O board*, which allows to test an ActivVisionTools application before connecting it to the process. This virtual board can also be used to create an interface to a yet unsupported digital I/O board via the programming interface of ActivDigitalIO (see section 3.2 on page 42 for details).

### What is Digital I/O?

The term “digital I/O” is somewhat misleading, because from the point of view of ActivVisionTools all input and output is performed in digital form, be it via ActivDigitalIO, ActivSerial, or ActivFile (or via the programming interface). Instead, the main difference between these tools (besides the fact that ActivSerial and ActivFile are pure output tools) lies in the content of the transmitted data: Using ActivSerial or ActivFile (or the programming interface) you can transmit *complex data* like measuring results or read bar codes, represented by one or more *characters*. In contrast, ActivDigitalIO transmits “only” *binary (logical) data*, i.e., the values `'TRUE'`or `'FALSE'`, for each channel independently. Logical data is well-suited for the output of evaluations (e.g., “part is okay”), to signal events (e.g., “next part to inspect arrived, grab image now”), or to signal the state of an application (e.g., “inspection results available via digital output”).

As already noted, digital I/O boards provide multiple input and output channels. These channels

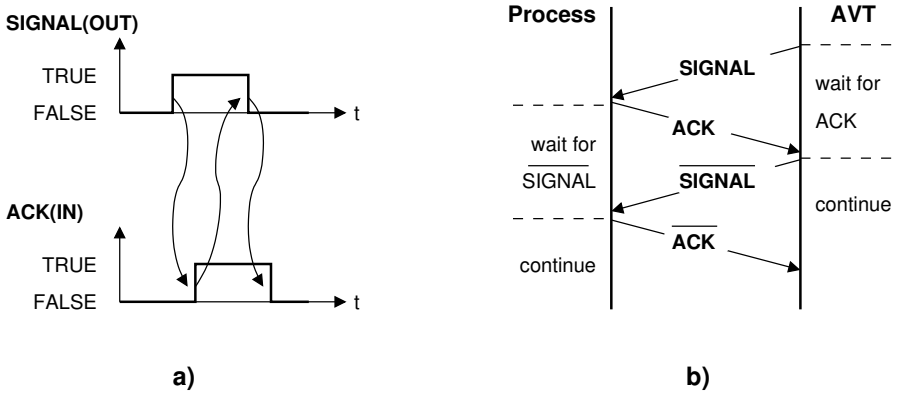



Figure 1.1: Example for using a handshake: a) timing diagram, b) communication diagram.



can be read and written to independently. Besides, the channels are divided into so-called *ports*, each consisting of 8 channels. This allows to access multiple channels simultaneously.

When dealing with binary signals, one must take care to distinguish their physical value, which can be "low" or "high", from their logical value `'TRUE'`(also called "set") and `'FALSE'`(also called "cleared"). Correspondingly, ActivDigitalIO lets you choose for each channel whether it is `'TRUE'` when the signal is "low" or "high", respectively.

### Synchronization Mechanisms

When connecting an ActivVisionTools application to the control system of a process via digital I/O, the communication between those two partners must be *synchronized* at some points. Some synchronization is performed automatically, e.g., if the ActivVisionTools application waits until a certain input channel becomes `'TRUE'` before grabbing an image and processing it. However, if the ActivVisionTools application is to output the evaluation of an inspected part, the process control cannot wait for this signal to become `'TRUE'`. In such a case you must synchronize the two partners, e.g., by setting another output channel to `'TRUE'` when the output of the evaluation result is valid.

Another possible reason for explicit synchronization is that one partner wants to be sure that the other has "received" a signal before it can continue itself. For example, an ActivVisionTools application might want to assure that the process has received the evaluation of the current part before continuing to inspect the next one. A synchronization mechanism well-suited to this case is the so-called *handshake*: Upon receiving the signal, the receiving partner sets another channel, typically called "Acknowledge" or "ACK" for short, to `'TRUE'`. The sender of the original signal waits for this acknowledgment; after receiving it, the original signal is cleared (denoted by $\overline{\text{SIGNAL}}$), followed by the acknowledgement ($\overline{\text{ACK}}$). This sequence is depicted in

figure 1.1, both as a timing diagram and a communication diagram.

### Linking Digital I/O Channels to an ActivVisionTools Application

Besides performing the low-level interactions with the digital I/O board, ActivDigitalIO provides high-level mechanisms that facilitate linking channels to your ActivVisionTools application. For example, you can interactively assign individual evaluations of tools, ROIs, objects, or features (called *evaluation objects*) to output channels.

Furthermore, ActivDigitalIO already predefines a set of output signals that describe the current state of the ActivVisionTools execution cycle, e.g., `'Grabbing Image'`, `'Processing Image'`, or `'Results Available'`, and a set of input signals that allow to synchronize process and ActivVisionTools application, e.g., `'Start Execution'`, `'Allow Grabbing'`, or `'Stop Execution'` (see section 1.3 on page 6 for detailed information). These signals (also called *action objects*) can be assigned interactively to individual channels like the evaluations.

Via the programming interface, you can also create your own action objects for input and output signals and integrate them into the ActivVisionTools execution cycle (see section 3.1 on page 26 for details).

## 1.2 The Sub-Tools of ActivDigitalIO

ActivDigitalIO provides a *master tool*, a *support tool*, and some dialogs. In figure 1.2, they are depicted together with other ActivVisionTools in an example blob analysis application.

`AVTDIOBoard` is the *master tool* of ActivDigitalIO. It allows to specify the employed digital I/O board and to enable the input and output. How to use `AVTDIOBoard` is described in more detail in chapter 2 on page 13.

Via `AVTDIOBoard`, you can open a dialog that allows to configure the `Board Settings`, e.g., the number of ports or whether to use handshaking. How to use this dialog is described in section 2.1 on page 14.

`AVTDIOBoardStatus` is a *support tool* of ActivDigitalIO. It displays the current state of the input and output channels; besides, it allows to set input and output values interactively. How to use `AVTDIOBoardStatus` is described in chapter 2 on page 13.

Via `AVTDIOBoard`, you can open other dialogs that allow to assign signal objects to input and output channels and to configure the corresponding parameters, e.g., timeout values. How to use these dialogs is described in section 2.2 on page 16.

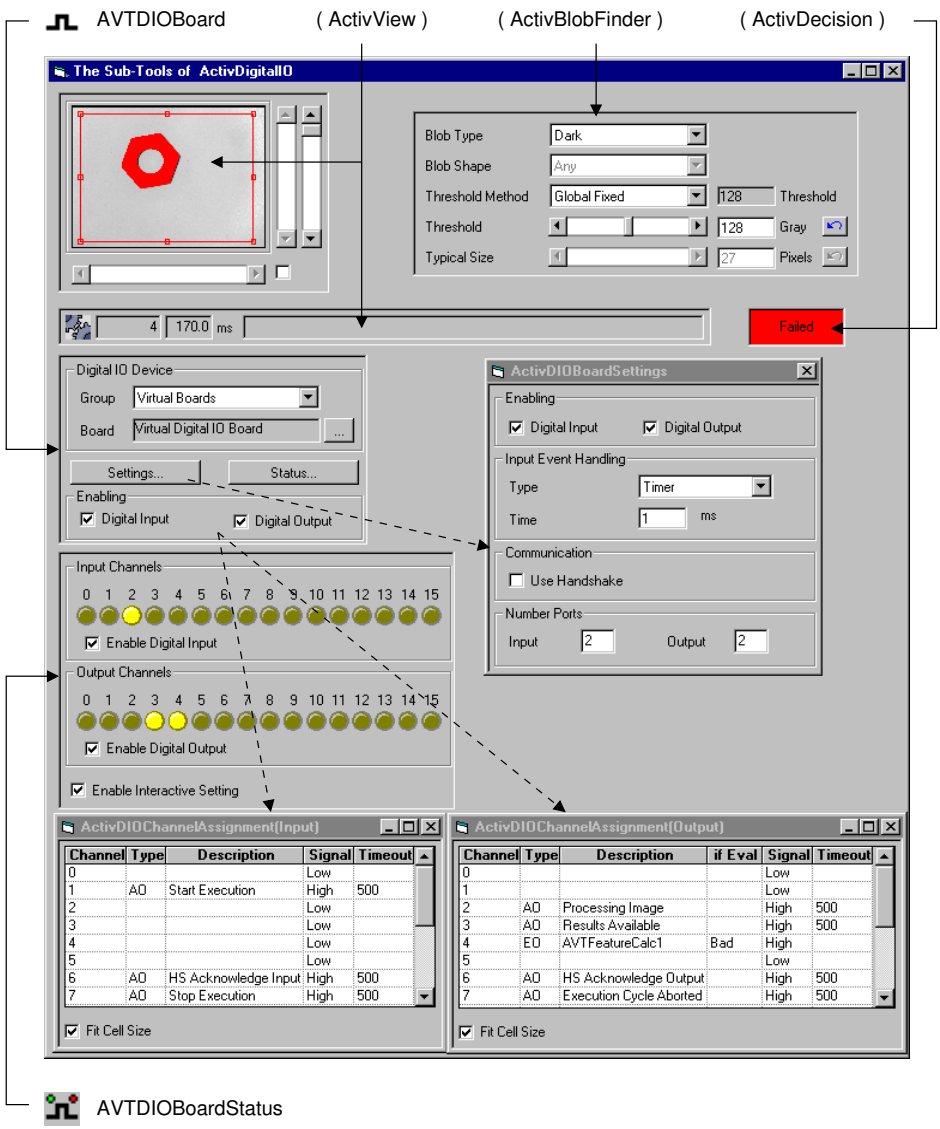



Figure 1.2: The sub-tools of ActivDigitalIO in a blob analysis application.

## 1.3 A Closer Look at the Interface of ActivVisionTools to Digital I/O

In this section, we take a closer look at the interface of ActivVisionTools to digital I/O. Section 1.3.1 gives an overview of the involved hardware and software components; section 1.3.2 on page 8 describes what digital I/O mechanisms are already predefined and integrated into the ActivVisionTools execution cycle. More information about the programming interface can be found in section 3.1 on page 26.

### 1.3.1 Components Involved in Digital I/O

To connect a process via digital I/O to an ActivVisionTools application, many components have to interact and cooperate (see figure 1.3).

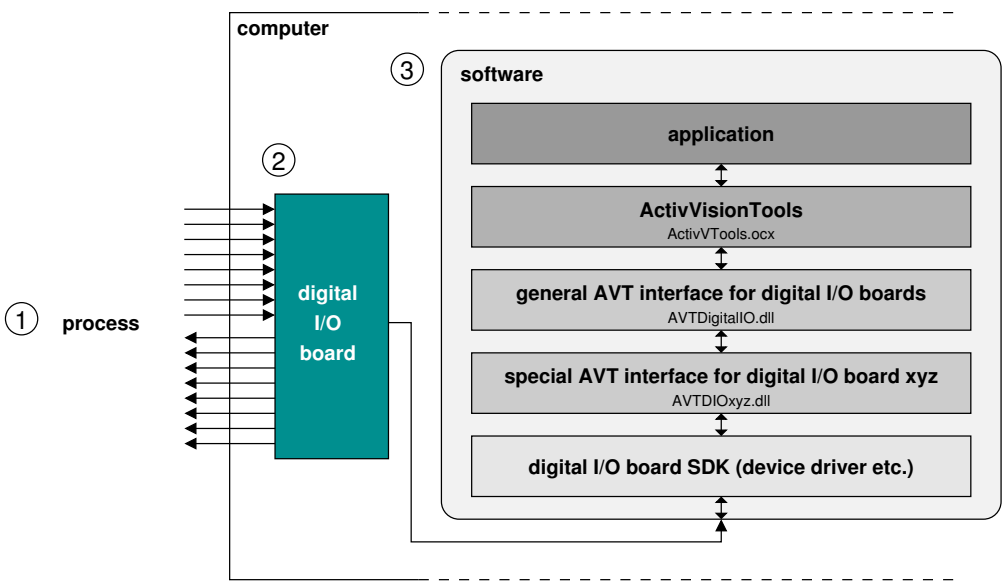



Figure 1.3: Components involved in digital I/O.

As already mentioned, the process is connected to the computer via a so-called digital I/O board, which provides multiple input and output channels. On the software side, the digital I/O board is accessed through a hierarchy of components: The low-level interaction with the board, i.e., reading input and writing to output channels, is performed by the so-called SDK (software development kit) provided by the manufacturer of the digital I/O board, which typically consists of a device driver and additional software facilitating the access.

ActivVisionTools does not use the SDK directly but through two layers of interfaces. The reason is that even if the functionality of digital I/O boards does not vary much over the different types and make, there is no common software interface. Therefore, for each supported board group ActivVisionTools provides a special interface, which "translates" the commands of the SDK for accessing the board into a set of commands common to all digital I/O board interfaces.

The next software layer, the so-called general ActivVisionTools interface for digital I/O boards, is the heart of digital I/O from the point of view of ActivVisionTools: In an own thread, it checks the current state of the input channels (based on interrupts or polling, depending on the used board), and reports changes to the main ActivVisionTools thread. In the other direction, the ActivVisionTools thread can tell the interface thread to set or clear certain output channels.

As with other functionality, ActivVisionTools itself encapsulates digital I/O in a set of controls and classes, which let you configure your application quickly with a few mouse clicks, but also allow to integrate more complex functionality via the programming interface.

### 1.3.2 Predefined Digital I/O in ActivVisionTools

#### Synchronizing and Describing the State of the Execution Cycle

ActivVisionTools already provides predefined mechanisms that allow to link digital input or output to important activities in the execution cycle (see figure 1.4). For example, if you assign an input channel to the predefined input signal (i.e., action object) `'Allow Grabbing'`, ActivVisionTools automatically checks whether this signal is set before grabbing the next image; if it isn't, they wait until the signal is set (or a timeout occurs). Similarly, you can synchronize the start of image processing by assigning a channel to the input signal `'Allow Image Processing'`, or the start of the output by assigning a channel to the input signal `'Allow Result Output'`. If you don't assign channels to these signals, ActivVisionTools doesn't wait for them to be set, of course.

Furthermore, you can inform the process of the current "phase" or state of the ActivVisionTools execution cycle by assigning output channels to the signals (i.e., action objects) `'Grabbing Image'`, `'Processing Image'`, or `'Writing Results'`: These signals are then set when the corresponding activity starts and cleared upon finishing. The predefined output signal `'Results Available'`, finally, is set after all output tools have been executed and cleared at the beginning of the next cycle (if it is assigned to an output channel). It can therefore be used to inform the process that output channels assigned to evaluations contain valid values.

More information about the ActivVisionTools execution cycle can be found in the Advanced User's Guide for ActivVisionTools in chapter 3 on page 53; section 3.6.2.1 on page 103 describes which predefined action objects are added if you use multiple instances of ActivView.

#### Starting, Stopping, and Aborting an Execution Cycle

The signals (action objects) described above are linked firmly to the ActivVisionTools execution cycle, i.e., they are waited for only at certain moments or set during a certain phase. In contrast, the following predefined signals can be set at arbitrary moments during the execution cycle. Again, you must assign channels to the signals to "activate" them.

- `'Execution Cycle Aborted'` (output):

  If the ActivVisionTools application encounters a timeout while waiting for a signal (or for a handshake signal, see below) or if the grabbing fails, the execution cycle is aborted immediately, and the output signal `'Execution Cycle Aborted'` is set.

  Note that the application is not stopped explicitly; i.e., if the application is running continuously when encountering a timeout, the current cycle is aborted and the next one is started automatically.

  The signal is cleared automatically at the beginning of a new execution cycle (see figure 1.4).

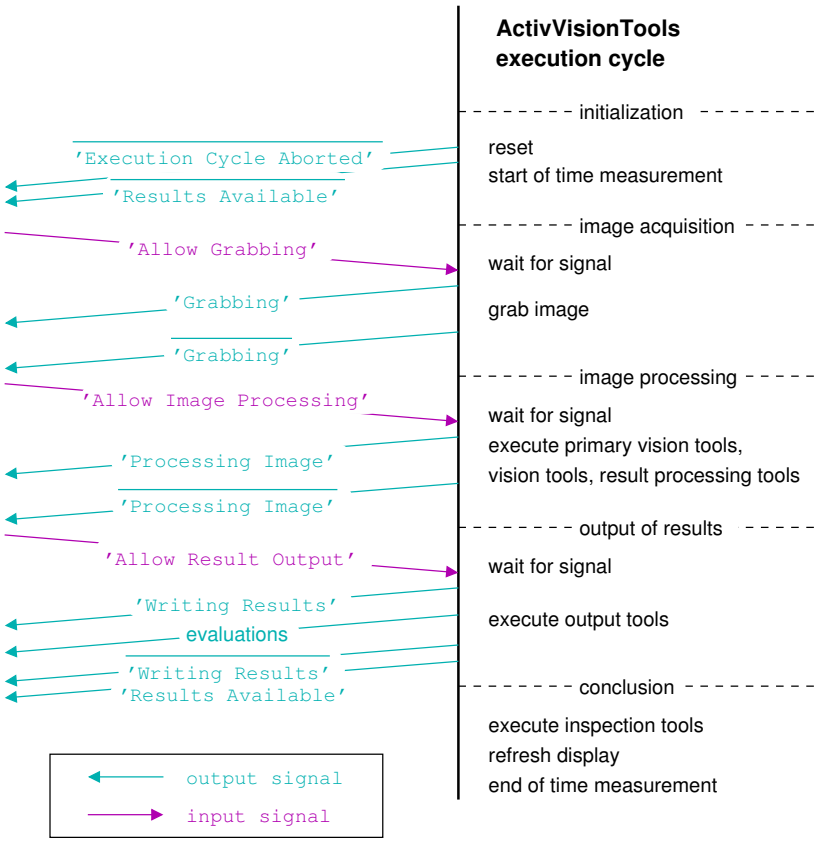



Figure 1.4: Input and output signals in the ActivVisionTools execution cycle.

- `'Start Execution'` (input):

  When this input signal is set, the ActivVisionTools application immediately starts to execute continuously. If the application is already running, the signal has no effect.

- `'Stop Execution'` (input):

  When this input signal is set, the ActivVisionTools application finishes the current execution cycle and then stops. If the application is not running, the signal has no effect.

- `'Abort Execution Cycle'` (input):

  When this input signal is set, the ActivVisionTools application aborts the current execution cycle at the next possible moment, i.e., not in the middle of executing a tool but directly afterwards. Furthermore, it sets the output signal `'Execution Cycle Aborted'` described above.

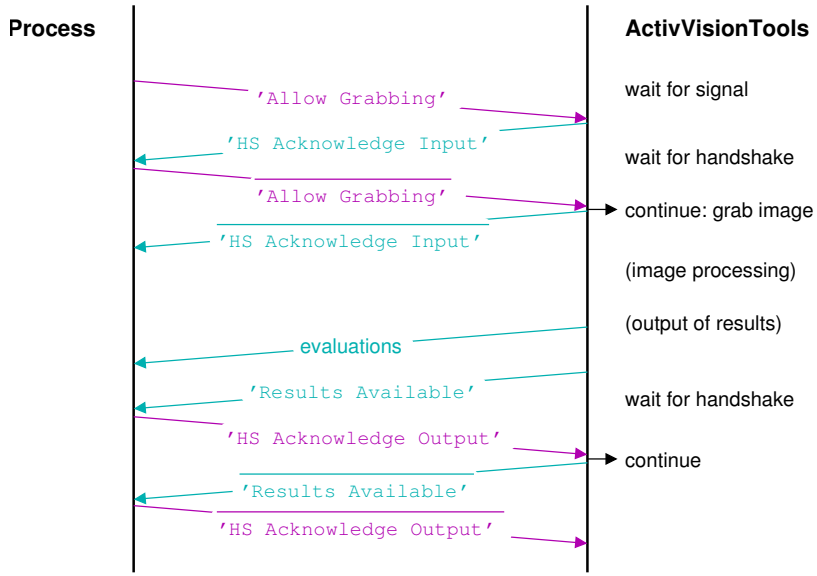



Figure 1.5: Example application using handshakes; besides some evaluations, only the signals `'Allow Grabbing'` and `'Results Available'` are used (i.e., assigned to channels)..

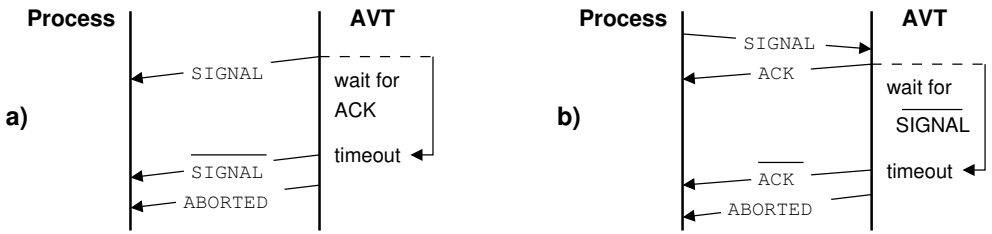



Figure 1.6: Timeouts during handshaking: a) output, b) input.

Note that this input signal does not stop the ActivVisionTools application, i.e., if the application was running continuously when receiving the signal `'Abort Execution Cycle'` it will abort the current cycle and immediately start a new one. You can stop the application from continuing by setting the signal `'Stop Execution'` (see above) in addition to `'Abort Execution Cycle'`.

### Using Handshake Signals

Section 1.1 on page 2 already described how so-called handshakes can be used to synchronize process and ActivVisionTools application. The main principle is that the partner receiving a signal immediately "acknowledges" it by setting another signal. The sending partner then clears the original signal, upon which the acknowledging signal can be cleared as well (see figure 1.1 on page 3). For this, ActivVisionTools provides two predefined signals (action objects):

- `'HS Acknowledge Output'`:

  If handshaking is enabled, this signal must be assigned to an input channel. The process then must use this channel to acknowledge output signals, i.e., signals set by the ActivVisionTools application. The application waits until `'HS Acknowledge Output'` is set or until a timeout occurs. If the handshake arrives in time, the application clears the output signal and continues. In figure 1.5 on page 10, this is depicted for the signal `'Results Available'`. Note that no handshake is performed for evaluation objects.

  If a timeout occurs, the application clears the output signal and sets the signal `'Execution Cycle Aborted'` as depicted in figure 1.6a on page 10.

  Note that when using handshakes, output signals like `'Results Available'` do not describe the state of the ActivVisionTools application anymore, as their clearing is part of the handshake!

- `'HS Acknowledge Input'`:

  If handshaking is enabled, this signal must be assigned to an output channel. When the ActivVisionTools application receives one of the input signals described in previous sections, it sets the signal `'HS Acknowledge Input'`. Then, it waits until the input signal is cleared or until a timeout occurs. If the handshake arrives in time, the application clears the signal `'HS Acknowledge Input'` and continues. In figure 1.5 on page 10, this is depicted for the signal `'Allow Grabbing'`. Note how the grabbing only starts after the handshake is completed.

  If a timeout occurs, the application clears the signal `'HS Acknowledge Input'` and sets the signal `'Execution Cycle Aborted'` as depicted in figure 1.6b on page 10.

  Note that no handshake is performed for the input signal `'Abort Execution Cycle'`.

## 1.3.3 When are Input Channels Checked?

The task of keeping an ActivVisionTools application up-to-date on the current state of the input channels may seem trivial but it deserves a closer look. The main problem is that the application should not spend time unnnecessarily by checking the digital I/O board continuously just in case an input channel value changes, but on the other hand it is crucial that it does not "miss" a change, or notices it "too late".

ActivVisionTools approaches this problem by devoting an own thread to the task of communication with the digital I/O board, running in parallel to the meain ActivVisionTools thread, which performs the execution cycle and handles the user interaction (see also section 1.3.1 on page 6. Depending on the functionality provided by the digital I/O board (and by its SDK), the digital I/O thread can check the input channels via *polling* or based on *interrupts*. In the first mode, which is available for all boards, the thread checks (polls) the values of the input channels periodically, with a period which can be chosen by the user. In this mode, therefore, only signals that are applied for a shorter time than the length of the polling period are missed; this problem can be solved by assuring that the process does not clear signals too quickly.

Nevertheless, in the polling mode the thread still wastes time by checking the input channels unnecessarily often. The far more efficient mode is to let the board signal changes via an *interrupt*, which is then acted upon by the digital I/O thread. Unfortunately, not all digital I/O boards (or their SDKs, respectively) support this mode, neither does the virtual digital I/O board provided by ActivVisionTools.

Up to now, we discussed how the digital I/O thread keeps up-to-date on the current state of the input channels; what about the main thread, i.e., the ActivVisionTools application? The main point is that the application does not need to keep up-to-date continuously, but only at the moments when it needs to know the current value of an input channel or when it waits until an input signal is set. On these occasions, therefore, the main thread asks the digital I/O thread whether any of the input channels changed their value. This means that changing input channel values may show up at a later moment in the application than on the board itself, but nevertheless they will show up in time, i.e., when the application needs to know about them.

# Chapter 2

# Using ActivDigitalIO



This chapter will explain how to use ActivDigitalIO to synchronize the execution cycle of a machine vision application via digital input and to inform a process about its results via digital output.

The corresponding Visual Basic projects already contain a completely configured blob analysis application which checks nuts using ActivBlobFinder, ActivFeatureCalc, and ActivDecision. If you want to experiment with the projects in more detail we recommend to read the User's Manual for ActivBlobFinder, the User's Manual for ActivFeatureCalc, and the User's Manual for ActivDecision.

**2.1 Configuring the Connection to a Digital I/O Board . . . . . 14**
**2.2 Assigning Activities to Input and Output Channels . . . . . 16**
2.2.1 Assigning Evaluations To Output Channels . . . . . 16
2.2.2 Assigning Action Objects to Output Channels . . . . . 18
2.2.3 Assigning Action Objects to Input Channels . . . . . 20
2.2.4 Using Handshakes . . . . . 22

## 2.1 Configuring the Connection to a Digital I/O Board

AVTDIOBoard lets you setup and configure a connection to your digital I/O board quickly and easily. You can then test the connection interactively using AVTDIOBoardStatus.

**Visual Basic Example**

Preparation for the following example:

- ☐ Please open the project configuring\digital_configuring.vbp, which contains an already configured blob analysis application.
- ☐ Execute the application (Run ▷ Start or via the corresponding button). Load the image sequence misc\nuts1.seq.

The following steps are visualized in figure 2.1.

① First of all, you must specify the used digital I/O board. The combo box Group lets you select the board group. Currently, you can choose between 'Advantech Boards' and 'Virtual Boards'. With the button next to the text field Board you can select the actual board. If you selected the group 'Advantech Boards', the dialog depicted in the lower right corner of the figure pops up and displays the available boards, i.e., the boards installed in the computer. The group 'Virtual Boards' currently contains only a single instance.

② To enable the digital input and output check the boxes in the frame Enabling.

③ To further configure the connection, click the buttons [Settings] and [Status]. Two dialogs pop up.

④ In the dialog Board Settings, you can select the number of input and output ports in the frame Number Ports (for the virtual board). In the dialog Board Status, the number of channels is adapted accordingly.

⑤ In the frame Input Event Handling of the dialog Board Settings, you can choose the method for checking the values of the input channels (see section 1.3.3 on page 11). If you select 'None', the input channels are not checked at all, which has the same effect as disabling the digital input. If you select 'Timer', you can specify the length of the polling period in the text box Time.

⑥ To test the connection, check the box [✓] Enable Interactive Setting in the dialog Board Status. Now you can set and clear channels interactively by clicking on the corresponding symbols.

To experience the effect of polling the input channels, select a large value for the length of the polling period and then click on an input channel. It is marked as set after a

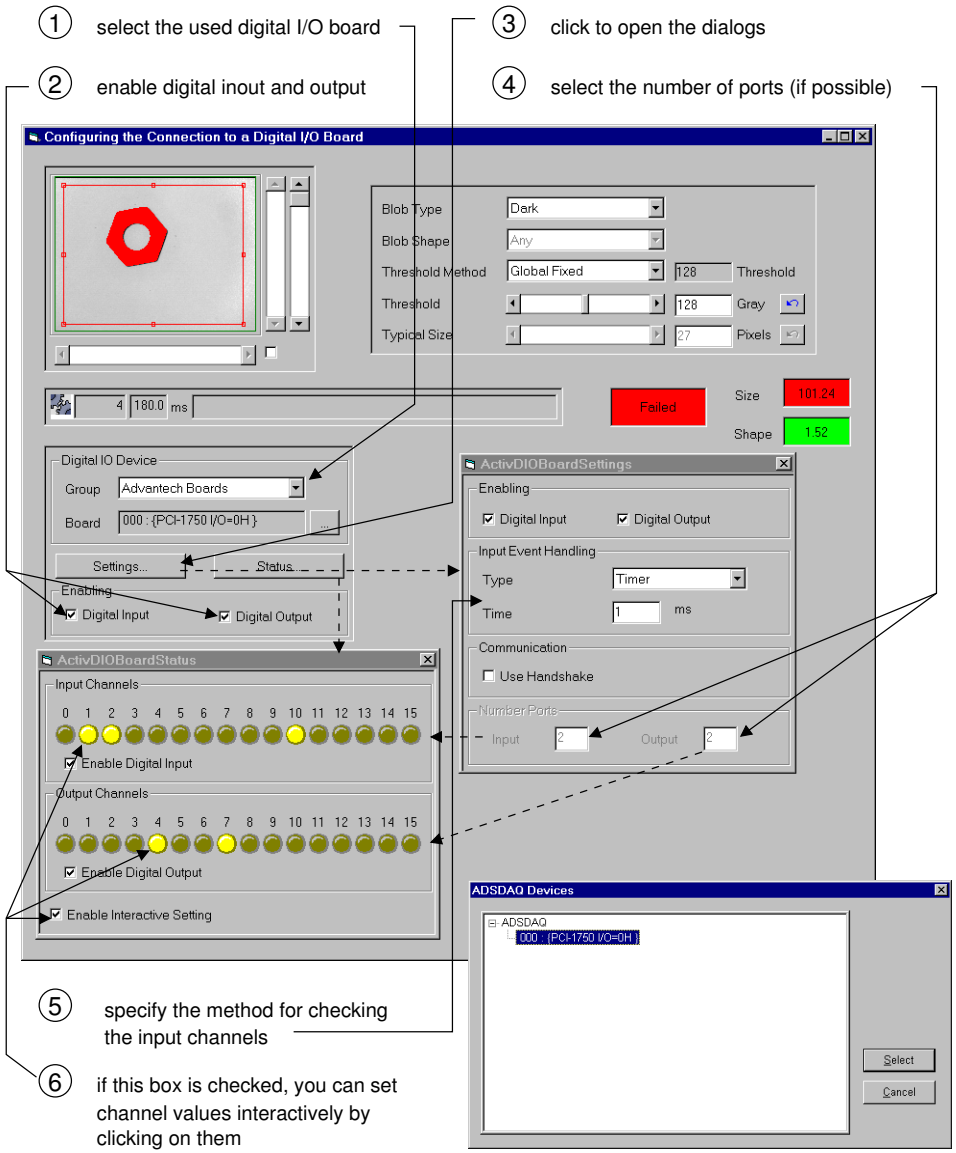



Figure 2.1: Configuring the connection to a digital I/O board.

considerable delay. How to use handshakes is explained in section 2.2.4 on page 22.

## 2.2 Assigning Activities to Input and Output Channels

The next step after configuring the connection is to assign activities to the input and output channels using `AVTDIOChannelAssignment`. The following sections each focus on one type of assignment. A special section is devoted to using handshakes. Section 3.1 on page 26 shows how to further customize the assignments via the programming interface.

**Visual Basic Example**

Preparation for the following example:

- ☐ If you worked on the previous example, you may continue using this project. Otherwise, open the project `assigning\digital_assigning.vbp`. This project is preconfigured for the virtual digital I/O board.
- ☐ Execute the application and load the image sequence `misc\nuts1.seq`. Click on `Status` in `AVTDIOBoard` to open the dialog `Board Status`.

### 2.2.1 Assigning Evaluations To Output Channels

First, we show how to assign evaluations to output channels. The following steps are visualized in figure 2.2.

1. Open the context menu of `AVTDIOBoard` or `AVTDIOBoardStatus` via a right mouse button click and select `Output Channel Assignment`. A dialog displaying the current assignments of the output channels appears.
2. To assign an evaluation to an output channel, click into its field in the column `Description`; in the appearing context menu, select `Insert Evaluation Object`. Another dialog pops up.
3. You can navigate the available evaluations by clicking on the `+` and `-` icons.
4. To assign an evaluation to an output channel, click on it with the right mouse button. In the appearing context menu you can select the channel for assignment. Note that only one object can be assigned to an output channel; if you assign an evaluation to an already used channel, the former assignment is replaced by the new one.

   You can remove an assignment explicitly via the context menus of both dialogs.
5. In the column `Signal`, you can specify for each assignment whether “setting” the signal corresponds to the physical signal value “low” or “high”. In the column `if Eval`, you can choose for which evaluation the signal is to be set.

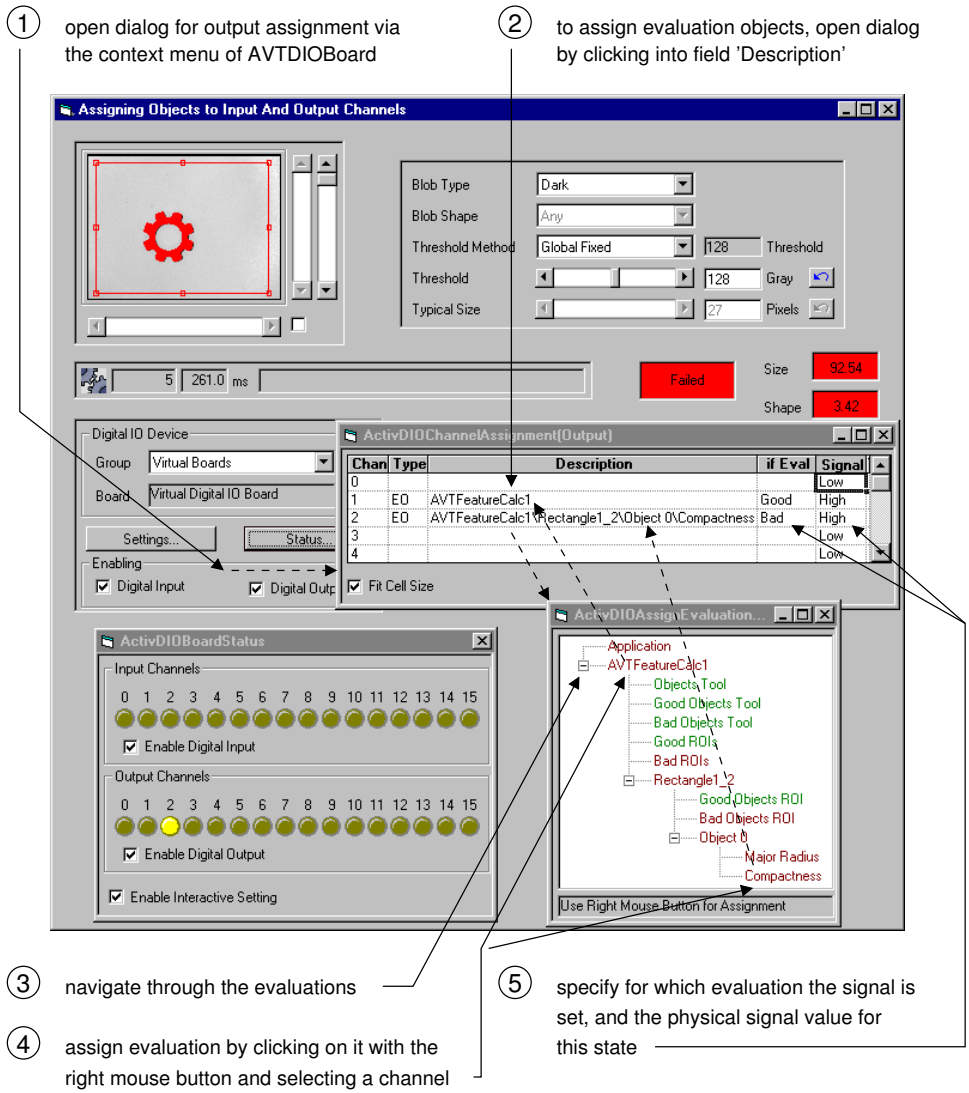



Figure 2.2: Assigning evaluations to output channels.

Now step through the image sequence and watch the resulting digital output in the dialog `Board Status`. Note that the tool tips of the channels show the assigned signals.

We continue the example on the next double page.

### 2.2.2 Assigning Action Objects to Output Channels

Action objects like `'Results Available'` can be assigned to output channels similar to evaluations. The following steps are visualized in figure 2.3 on page 19.

① To assign an action object to an output channel, click into its field in the column `Description`; in the appearing context menu, select `Insert Action Object`. Another dialog pops up, listing all available output action objects (see section 1.3.2 on page 8 for their description).

② To assign an action object to an output channel, click into its field in the column `Channel`. In the appearing context menu you can select the channel for assignment. Note that only one (action or evaluation) object can be assigned to an output channel; if you assign an action object to an already used channel, the former assignment is replaced by the new one.

You can remove an assignment explicitly via the context menus of both dialogs.

③ In the column `Signal`, you can specify for each assignment whether "setting" the signal corresponds to the physical signal value "low" or "high".

The content of the column `Timeout` is only relevant when using handshakes. Please refer to section 2.2.4 on page 22 for more information.

Now step through the image sequence and watch the resulting digital output in the dialog `Board Status`. Note that the tool tips of the channels show the assigned signals.

We continue the example on the next double page.

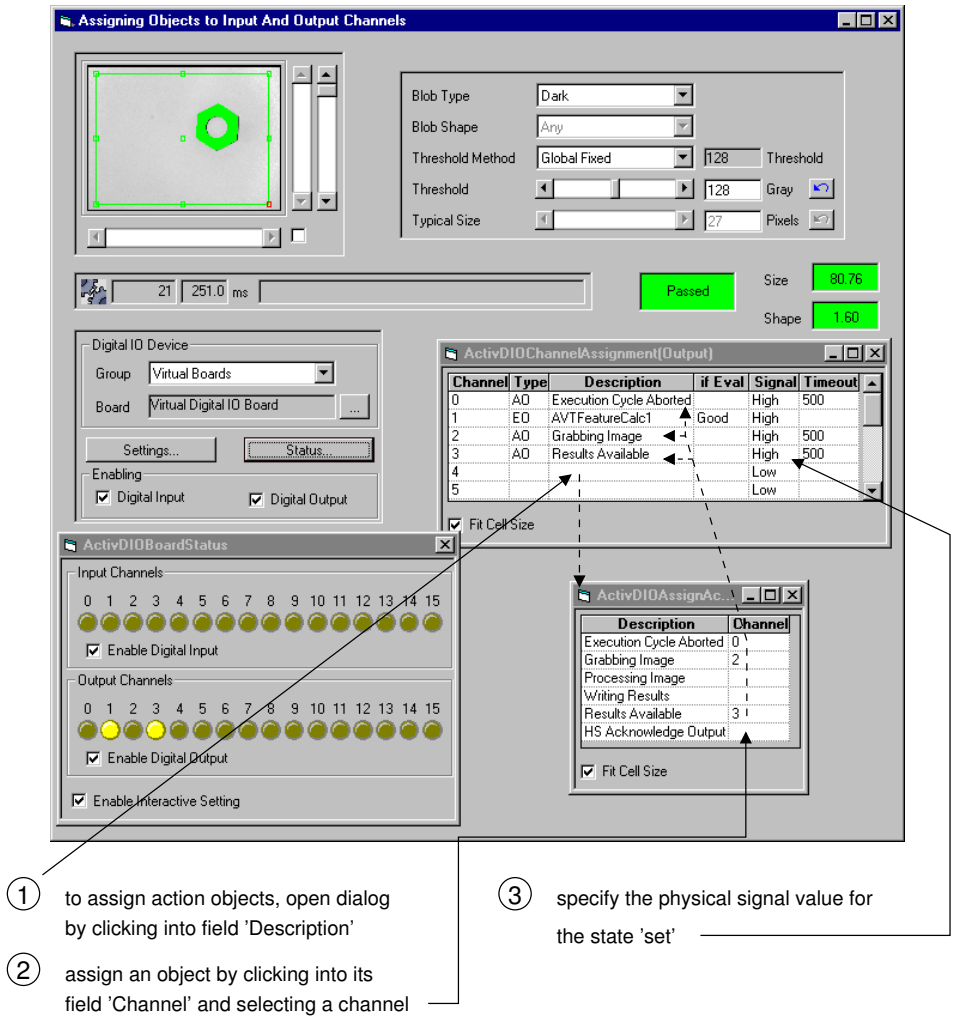



Figure 2.3: Assigning action objects to output channels.

### 2.2.3 Assigning Action Objects to Input Channels

To input channels, you can only assign action objects, e.g., `'Allow Grabbing'`. The following steps are visualized in figure 2.4.

1. Open the context menu of `AVTDIOBoard` or `AVTDIOBoardStatus` via a right mouse button click and select `Input Channel Assignment`. A dialog displaying the current assignments of the input channels appears.

2. To assign an action object to an input channel, click into its field in the column `Description`; in the appearing context menu, select `Insert Action Object`. Another dialog pops up, listing all available input action objects (see section 1.3.2 on page 8 for their description).

3. To assign an action object to an input channel, click into its field in the column `Channel`. In the appearing context menu you can select the channel for assignment. Note that only one object can be assigned to an input channel; if you assign an action object to an already used channel, the former assignment is replaced by the new one.

   You can remove an assignment explicitly via the context menus of both dialogs.

   ! Please note that **some digital I/O boards use certain input channels for signaling interrupts!** For example, the Advantech board PCI-1750 uses channels 0 and 8 for this purpose. Therefore, do not choose these channels for assignment when using interrupts for checking input channels.

4. In the column `Signal`, you can specify for each assignment whether "setting" the signal corresponds to the physical signal value "low" or "high".

5. In the column `Timeout`, you can specify a timeout value for the input signal (in milliseconds), i.e., how long the application is to wait for the signal to be set (see also section 1.3.2 on page 8).

   The timeout value is also used when waiting for a handshake. Please refer to section 2.2.4 on page 22 for more information.

In figure 2.4, we assigned the action objects `'Start Execution'`, `'Allow Grabbing'`, and `'Stop Execution'`. Therefore, you can now start and stop the application by setting the corresponding input channels interactively in the dialog `Board Status`. Note that the tool tips of the channels show the assigned signals. If you do not set the channel assigned to `'Allow Grabbing'`, a timeout occurs and is reported in `AVTViewStatus`. If you assigned an output channel to the action object `'Execution Cycle Aborted'` as depicted in figure 2.3 on page 19, the application will set this signal.

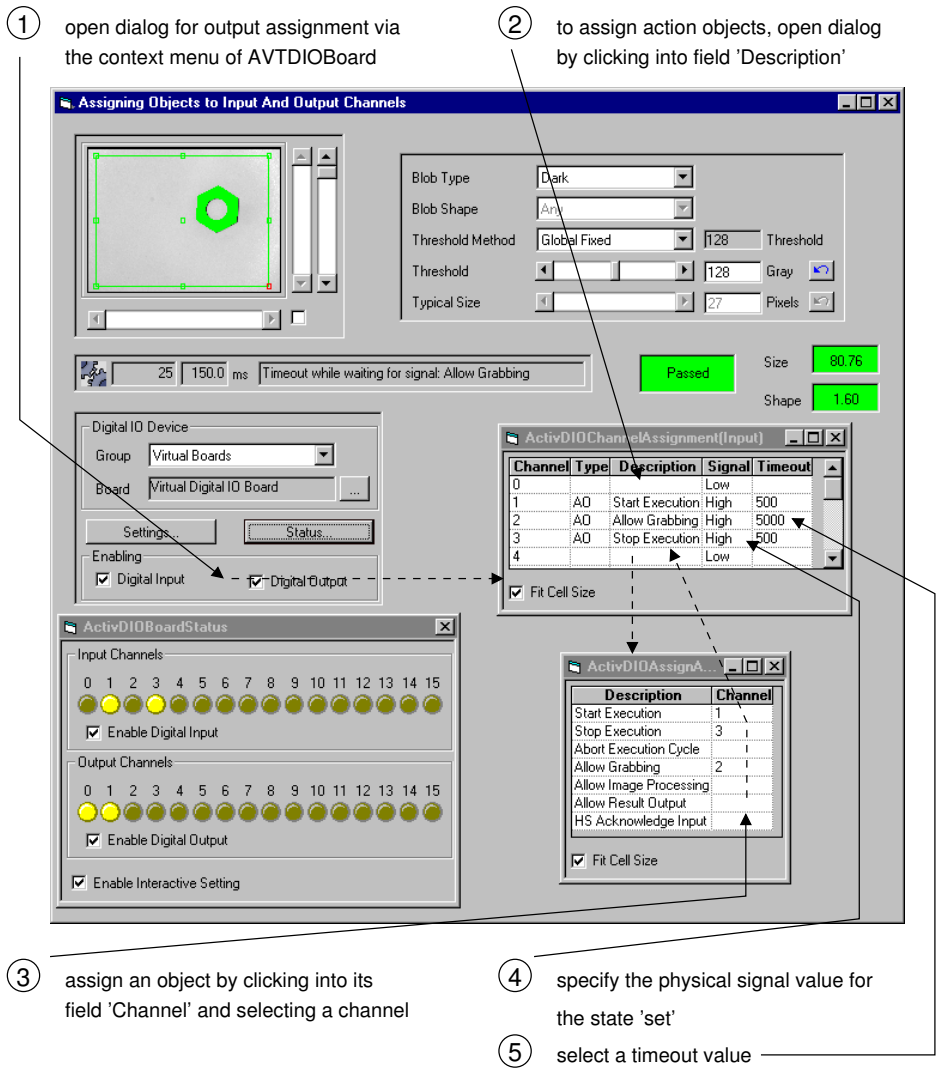



Figure 2.4: Assigning action objects to input channels.

### 2.2.4 Using Handshakes

To use handshakes as described in section 1.3.2 on page 11, you must select an input and an output channel for the signals of acknowledgement. The following steps are visualized in figure 2.5.

① Assign the action object `'HS Acknowledge Output'` to an input channel and the action object `'HS Acknowledge Input'` to an output channel as described in the previous sections.

② Now, the value selected in the column `Timeout` for input action objects is also used when the application waits for the input object (in the example: `'Allow Grabbing'`) to be cleared after setting the output handshake signal, i.e., `'HS Acknowledge Input'`.

For output objects, on the other hand, the value specified in the column `Timeout` determines how long the application waits for the input handshake signal, i.e., `'HS Acknowledge Output'`, to be set, after setting an output action object (in the example: `'Results Available'`).

③ Finally, check the box ☑ `Use Handshake` in the dialog `Board Settings` to enable the handshake.

Please note that if you enable handshaking, handshakes are expected for all predefined action objects that are assigned to input or output channels; only evaluation objects are spared. In the example in figure 2.5, we therefore assigned only a few action objects to facilitate experiments with handshakes.

A good introduction to the effect of handshakes is to "re-enact" the digital I/O depicted in figure 1.5 on page 10: Start a single execution cycle by grabbing the next image of the image sequence. Then, set the input signal `'Allow Grabbing'`; the application acknowledges this signal automatically by setting the output signal `'HS Acknowledge Input'`. In return, you must now clear the signal `'Allow Grabbing'`, after which the application clears `'HS Acknowledge Input'`, grabs the image and processes it. Depending on the content of the image, the application then signals the evaluation and then sets the output signal `'Results Available'`. Now, its your turn to acknowledge this signal by setting the signal `'HS Acknowledge Output'`, after which the application clears the signal `'Results Available'`, and you can clear `'HS Acknowledge Output'`.

In figure 2.5, a timeout occurred while the application waited for the acknowledgment of `'Results Available'`, as can be seen in `AVTViewStatus`.

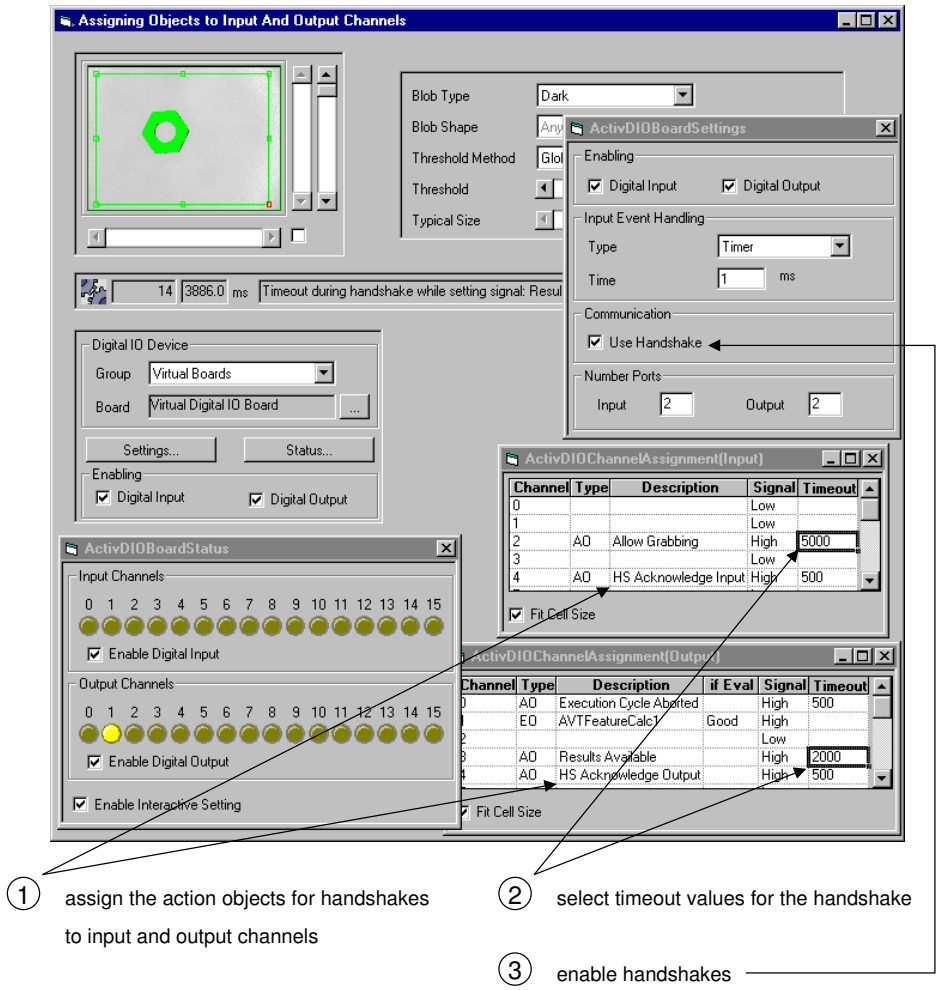



Figure 2.5: Using handshakes.

# Chapter 3

# Tips & Tricks



This chapter contains additional information about ActivDigitalIO, e.g., how to use the programming interface of ActivVisionTools to create your own signals (i.e., action objects) for digital I/O.

**3.1 Programming Interface . . . . . 26**
3.1.1 Basic Principles . . . . . 26
3.1.2 Example . . . . . 32
**3.2 How to Connect to Unsupported Digital I/O Boards . . . . . 42**

## 3.1 Programming Interface

The previous chapters and sections showed how to use the ActivDigitalIO interactively, i.e., via the graphical user interfaces presented by the underlying ActiveX controls. In this mode, you can configure the digital input and output rapidly and easily, without any programming. However, there is more to ActivVisionTools than the graphical user interfaces: Because ActivVisionTools comes as a set of ActiveX controls, it provides you with an open programming interface, thereby offering full flexibility.

In this section, we show how to use the programming interface to create your own signals (i.e., action objects) for digital input and output and how to integrate them into an ActivVisionTools application. Please note that we assume that readers of this part have at least a basic knowledge of Visual Basic.

! To work with the programming interface, you must first **add the ActivVisionTools type library to the project's references** by checking the box labeled `ActivVisionTools Type Library` in the Visual Basic menu dialog `Project` ▷ `References`.

### 3.1.1 Basic Principles

#### Creating Your Own Action Objects

The first step towards creating your own action objects is to declare variables of the class `AVTDIOActionObjectClass` as follows:

```
Private WithEvents atMyInputActionObject As AVTDIOActionObjectClass
Private WithEvents atMyOutputActionObject As AVTDIOActionObjectClass
```

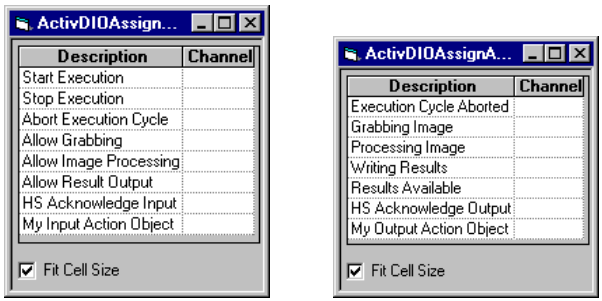
All action objects are stored in a so-called *global object repository*. To insert your own action objects, you therefore need access to this repository. Thus, declare a corresponding variable as follows:

```
Private atGlobalObjRep As AVTDIOGObjectRepositoryC
```

In the event handler `Form_Load`, i.e., after the ActivVisionTools have been loaded, you can then create the action objects and insert them into the repository. First, you initialize the variable pointing to the global object repository:

```
Private Sub Form_Load()
  Set atGlobalObjRep = New AVTDIOGObjectRepositoryC
```

Input action objects are created in three steps: First, the variable is initialized. Then, you specify a string describing the input object. This string is afterwards displayed in the list of input action




a) b)

Figure 3.1: Lists of action objects with newly created: a) input object, b) output object.

objects available for assignment (see figure 3.1a). Finally, you insert the object into the repository via its method `InsertInputObjectIfNew`; the first parameter of this method acts as an identifier for the action object in the repository.

```
  Set atMyInputActionObject = New AVTDIOActionObjectClass
  atMyInputActionObject.Description = "My Input Action Object"
  Set atMyInputActionObject = _
    atGlobalObjRep.InsertInputObjectIfNew("MyIAO", atMyInputActionObject)
```

Output action objects are created similary; the only difference is that you use the method `InsertOutputObjectIfNew` to insert them into the repository instead:

```
  Set atMyOutputActionObject = New AVTDIOActionObjectClass
  atMyOutputActionObject.Description = "My Output Action Object"
  Set atMyOutputActionObject = _
    atGlobalObjRep.InsertOutputObjectIfNew("MyOAO", atMyOutputActionObject)
End Sub   ' Form_Load
```

You can also access predefined action objects via functions of `AVTDIOBoard`. The following code creates a reference to the output action object `'Execution Cycle Aborted'`:

```
Private WithEvents atOAOAborted As AVTDIOActionObjectClass

Private Sub Form_Load()
  Set atOAOAborted = AVTDIOBoard1.AOOutputAbortCycle
End Sub   ' Form_Load
```

### Setting Signals and Waiting for Them

Within your application, you can now easily set your own output signals:

```
Private Sub CommandSet_Click()

bSuccessfully = atMyOutputActionObject.SetSignal(True)
```

The return value shows whether the signal could be set successfully. If handshakes are enabled, the method `SetSignal` automatically performs the necessary handshake actions, i.e., it waits for the input handshake signal `'HS Acknowledge Output'` and then clears the output signal. If a timeout occurred while waiting for `'HS Acknowledge Output'`, `SetSignal` returns `FALSE` (but clears the output signal nevertheless).

You can clear output signals with the same method; as no handshake is performed when clearing signals, it is rarely necessary to access the return value.

```
Call atMyOutputActionObject.SetSignal(False)
```

You can check whether an input signal is currently set using the method `IsSignaled`:

```
Dim bIsSet As Boolean

bIsSet = atMyInputActionObject.IsSignaled
```

Note that this method only queries the current value of the input channel. It neither waits for the signal to be set, nor performs it any handshake actions, even if handshakes are enabled.

In contrast, the method `IsSignaledWait` waits for an input channel to be set, with the return value showing whether the waiting was successful:

```
Dim bIsSet As Boolean
Dim atCommCode As AVTCommErrorCodesEnum

bIsSet = atMyInputActionObject.IsSignaledWait(atCommCode)
```

If handshakes are enabled, the method automatically performs the necessary handshake actions, i.e., it sets the output signal HS Acknowledge Input, waits for the input signal to be cleared and then clears HS Acknowledge Input. In this case, `IsSignaledWait` can be unsuccessful because of two reasons: First, because a timeout occurred before the input signal was set, and secondly, because a timeout occurred while waiting for the handshake. The (optional) parameter of `IsSignaledWait` can be used to differentiate these two cases:

```
If atCommCode = eCommTimeout Then
  ' timeout
ElseIf atCommCode = eCommHSTimeout Then
  ' handshake timeout
End If
```

### Reacting to Changing Input Values

The methods described above read and wait for signals explicitly, i.e., at the moment the line of code is reached. Via the event `ValueChanged`, you can "hook" your application to changes of an input or output signal. This is especially useful in connection with predefined signals. For example, if you create a reference to the output action object `'Execution Cycle Aborted'`

```
Private WithEvents atOAOAborted As AVTDIOActionObjectClass

Private Sub Form_Load()
  Set atOAOAborted = AVTDIOBoard1.AOOutputAbortCycle
End Sub   ' Form_Load
```

you can then execute some code whenever `'Execution Cycle Aborted'` is set by creating an event handler for `ValueChanged` like this:

```
Private Sub atOAOAborted_ValueChanged(ByVal lNewValue As Long)

  If atOAOAborted.IsSignaled = True Then
    (reaction)
  End If

End Sub
```

Please note that the event is not raised if you modify action objects interactively via `AVTDIO-BoardStatus`!

### Useful Events in the ActivVisionTools Execution Cycle

The previous sections described the methods provided by the programming interface to set output and read input channels; this section addresses the question how to integrate calls to these methods into the ActivVisionTools execution cycle.

The ActivVisionTools are executed automatically in a suitable sequence, without any action being necessary from your part. However, they allow to integrate code into the execution cycle by raising a set of events, to which you can attach your own event handlers. Figure 3.2 shows at which moments these events are raised, in relation to the predefined digital input and output signals (see also section 1.3.2 on page 8).

- `CycleStart` (`AVTView`)

  ```
  Private Sub AVTView1_CycleStart()
  ```

  This event is raised by `AVTView` at the beginning of each execution cycle.

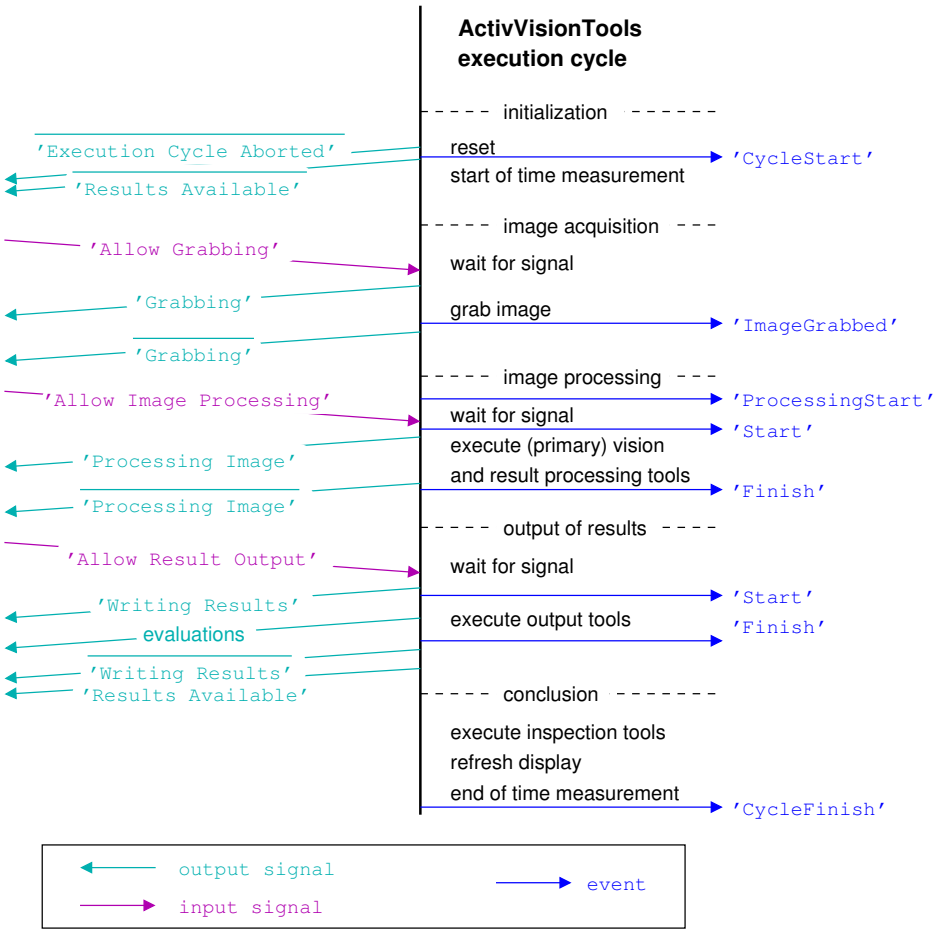



Figure 3.2: Input and output signals and events in the ActivVisionTools execution cycle.

- `ImageGrabbed (AVTView)`

```
Private Sub AVTView1_ImageGrabbed(hImage As HALCONXLib.HUntypedObjectX, _
                                  ByVal lError As Long)
```

This event is raised by `AVTView` after grabbing an image, passing the grabbed image as a parameter. In an event handler, you can, e.g., test whether the grabbing succeeded by testing whether the image is not `Nothing`. Please note that when creating a handler for this event you must add the HALCON/COM type library to the project's references by checking the corresponding box in the Visual Basic menu dialog `Project` ▷ `References`.

- `ProcessingStart` (`AVTView`)

```
Private Sub AVTView1_ProcessingStart(atGlobalIconicData As _
                                     ActivVTools.AVTImageMessageClass)
```

  This event is raised by `AVTView` before the actual image processing starts, i.e., before the *primary vision tools* (like ActivAlignment) and *vision tools* (like ActivMeasure) are executed.

- `Start`, `Finish` (*vision tools*)

```
Private Sub AVTMeasure1_Start(atIconicData As _
                              ActivVTools.AVTImageMessageClass)

Private Sub AVTMeasure1_Finish(atToolResults As _
                               ACTIVVTOOLSLib.IAVTToolResult)
```

  The execution of each *vision tool* is "framed" by two events raised by the executed tool. Before starting to execute, a tool raises the event `Start`, passing its input data, i.e., the image and ROI(s) as a parameter. After its execution, i.e., its share of image processing, it raises the event `Finish`, now passing its results as a parameter. Note that while tools like ActivMeasure or ActivFeatureCalc calculate results in form of alphanumeric *features*, ActivBlobFinder yields *iconic results* in form of the extracted blob regions:

```
Private Sub AVTBlobFinder1_Finish(atIconicData As _
                                  ActivVTools.AVTImageMessageClass)
```

- `Start`, `Finish` (ActivDecision)

```
Private Sub AVTDecision1_Start(atIconicData As _
                               ActivVTools.AVTImageMessageClass, _
                              cToolResults As Collection)

Private Sub AVTDecision1_Finish(atToolResults As Collection)
```

  The execution of ActivDecision, too, is framed by two events. With the event `Start`, it passes its input data, consisting of the input image and the collection of the results of the *vision tools*. After executing, i.e., evaluation the tool results, ActivDecision raises the event `Finish`, again passing the collection of tool results, which now also contain the evaluations.

- `Start`, `Finish` (*output tools*)

```
Private Sub AVTDIOBoard1_Start(atIconicData As _
                               ActivVTools.AVTImageMessageClass, _
                              cToolResults As Collection)

Private Sub AVTDIOBoard1_Finish(cToolResults As Collection)
```

  Output tools like ActivDigitalIO also frame their execution by raising two events. With the

event `Start`, they pass their input data, consisting of the input image and the collection of tool results, including their evaluations. After executing, which in case of ActivDigitalIO means the output of assigned evaluation objects, the event `Finish` is raised, again with the (unchanged) collection of tool results.

- `CycleFinish` (`AVTView`)

```
Private Sub AVTView1_CycleFinish()
```

At the end of each execution cycle, `AVTView` raises the event `CycleFinish`.

### 3.1.2 Example

The ActivVisionTools distribution includes the (completely preconfigured) example Visual Basic project `programming\digital_programming.vbp`, which applies the methods described in the previous section to extend the nut inspection application described in chapter 2 on page 13. The following tasks are to be solved:

1. Upon starting the application, ActivVisionTools must first be initialized, without any digital I/O. After this initialization, an output signal is set to inform the process that ActivVisionTools is ready to process images.
2. The ActivVisionTools execution cycle is to be triggered by the process via the input signal `'Start Execution'`. If the process clears this signal, ActivVisionTools is to stop at the end of each execution cycle.
3. The process needs to know whether the grabbing succeeded. A successful grabbing is announced by the output signal `'Processing Image'`; to ensure that the process does not "miss" this signal, handshaking is used, but only for this signal. A failed grabbing is announced by the output signal `'Execution Cycle Aborted'`.
4. The ActivVisionTools application is to allow multiple "inspection programs", between which the process can choose via input signals. To put in concrete terms, the process specifies which type of nut (M8 or M10) is to be expected in the next image using two input signals; the ActivVisionTools application then adapt the evaluation parameters accordingly. As the values of these input signals are not known in advance by definition, an additional input signal is used to announce when these signals are "valid".
5. The ActivVisionTools application is to output the inspection result in a customized form: Beside the overall evaluation, two additional output signals are to be used to further describe the found object if it is evaluated as 'not okay' (other nut type or unknown object type). The output signal `'Results Available'` is used to inform the process that the inspection results are available.

Figure 3.3 depicts the corresponding communication protocol and the activities of ActivVisionTools and the process.

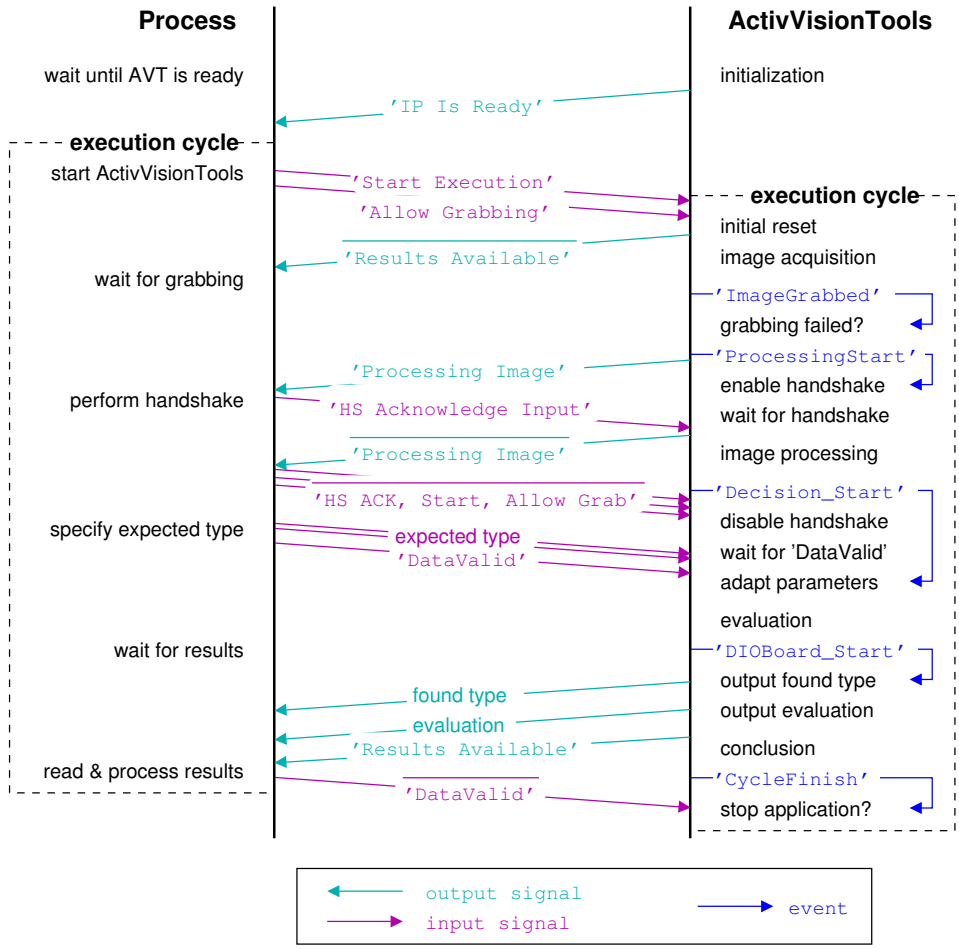



Figure 3.3: The communication protocol of the example application.

Figure 3.4 on page 34 shows the example project in action. The process is simulated in a separate form. Here is a list of the most important steps to experiment with the project:

1. First, initialize ActivVisionTools.
2. You can choose which type of nut is expected in the next image.
3. Start the next inspection cycle.

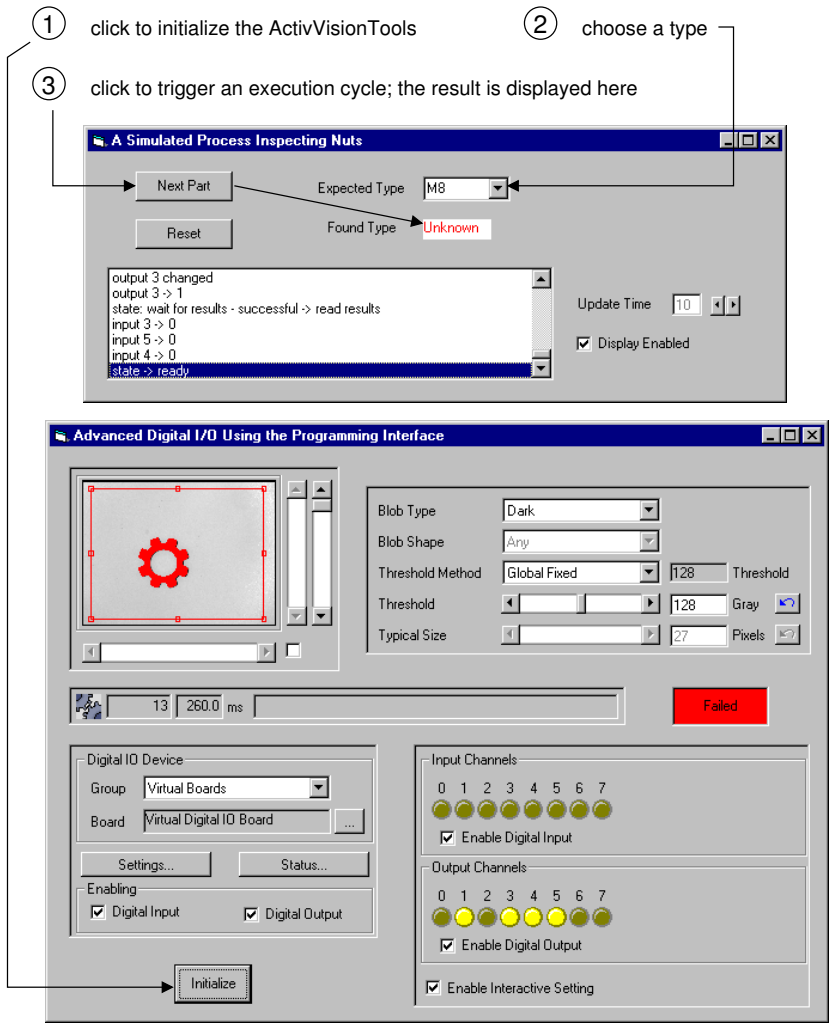
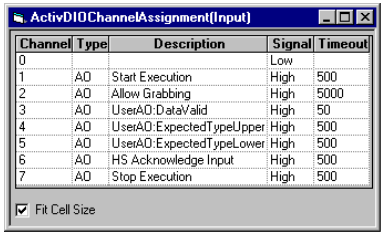
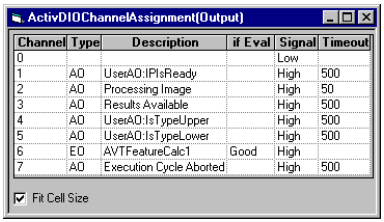



Figure 3.4: The example project in action.

In the following, we take a look at the Visual Basic code that solves the tasks described above. First of all, new action objects are created for input and output; their assignment is depicted in figure 3.5 on page 35. Most objects are self-explaning, only those specifying the expected and for the found type require some words of explanation: In both cases, two action objects are used, which can be interpreted as a two-bit code; therefore their names contain the ap-


ActivDIOChannelAssignment(Input)
Channel Type Description Signal Timeout
0 Low
1 AO Start Execution High 500
2 AO Allow Grabbing High 5000
3 AO UserAO:DataValid High 50
4 AO UserAO:ExpectedTypeUpper High 500
5 AO UserAO:ExpectedTypeLower High 500
6 AO HS Acknowledge Input High 500
7 AO Stop Execution High 500
Fit Cell Size

a)


ActivDIOChannelAssignment(Output)
Channel Type Description if Eval Signal Timeout
0 Low
1 AO UserAO:IPIsReady High 500
2 AO Processing Image High 50
3 AO Results Available High 500
4 AO UserAO:IsTypeUpper High 500
5 AO UserAO:IsTypeLower High 500
6 EO AVTFeatureCalc1 Good High
7 AO Execution Cycle Aborted High 500
Fit Cell Size

b)

Figure 3.5: Assignments in the example application: a) input channels, b) output channels.

pendix “Lower” or “Upper”. The combination `(Upper,Lower) = (0,1)` stands for the type M8, `(Upper,Lower) = (1,0)` for the type M10. The output signals also use the other two possible codes: `(Upper,Lower) = (1,1)` signals that the object is not a nut (which is checked via the feature `Compactness`), while `(Upper,Lower) = (0,0)` is used to signal that the expected type was found.

Besides the new action objects, some links to predefined action objects are created. Below, only the declaration of the objects is shown; they are created and integrated into the object repository in the event handler for `Form_Load` as described in section 3.1.1 on page 26.

```
’ the global object repository
Private atDIOGObjRepos As AVTDIOGObjectRepositoryC
’ own input action objects
Private WithEvents atDIOAODataValid As AVTDIOActionObjectClass
Private WithEvents atDIOAOExpectedTypeLower As AVTDIOActionObjectClass
Private WithEvents atDIOAOExpectedTypeUpper As AVTDIOActionObjectClass
’ access to predefined input action objects
Private WithEvents atDIOAOStartExec As AVTDIOActionObjectClass
Attribute atDIOAOStopExec.VB_VarHelpID = -1
’ own output action objects
Private WithEvents atDIOAOIPIsReady As AVTDIOActionObjectClass
Private WithEvents atDIOAOIsTypeLower As AVTDIOActionObjectClass
Private WithEvents atDIOAOIsTypeUpper As AVTDIOActionObjectClass
’ access to predefined output action objects
Private WithEvents atDIOAOAborted As AVTDIOActionObjectClass
```

The tasks described above are solved as follows:

1. **Initialize ActivVisionTools without digital I/O**
   Upon starting, ActivVisionTools is not ready yet, which is signaled by clearing the corresponding output signal in the event handler for `Form_Load`:

```
Call atDIOAOIPIsReady.SetSignal(False)
```

The initialization is performed in the following method, which is called when clicking the corresponding button. Note how the digital I/O is disabled before initializing the grabbing, i.e., grabbing the first image of the image sequence. Finally, the output signal that informs the process that ActivVisionTools is now ready for action.

```
Private Sub InitializeGrabbing()

  Dim atEventHandlingType As Integer

  ' disable digital I/O
  atEventHandlingType = AVTDIOBoard1.EventHandlingType
  AVTDIOBoard1.EnableDIOInput = False
  AVTDIOBoard1.EnableDIOOutput = False

  ' initialize the image processing
  Call AVTView1.InitGrabber

  ' enable digital I/O
  AVTDIOBoard1.EnableDIOInput = True
  AVTDIOBoard1.EventHandlingType = atEventHandlingType
  AVTDIOBoard1.EnableDIOOutput = True

  bIPIsInitialized = True

  ' inform process via digital output channel
  Call atDIOAOIPIsReady.SetSignal(True)
End Sub
```

2. **Trigger execution cycle by process**
   By assigning the input signal `'Start Execution'` to a channel, the process can automatically trigger an execution cycle by setting this signal. However, by default ActivVisionTools would now run continuously. To stop it at the end of the cycle, the following code is inserted into the handler for the event `CycleFinish`:

```
Private Sub AVTView1_CycleFinish()

  If atDIOAOStartExec.IsSignaled = False Then
    AVTView1.RunState = False
  End If

End Sub
```

Thus, the ActivVisionTools stop if the signal `'Start Execution'` is cleared at the time the event handler is reached.

3. **Synchronize and check grabbing**
   The start of the grabbing is synchronized via the predefined input signal `'Allow Grabbing'`. Then, the process waits for the output signal `'Processing Image'`, which implicitely means that the grabbing succeeded. If the grabbing fails, ActivVisionTools sets the signal `'Execution Cycle Aborted'` instead.

   In the example project, a grabbing failure occurs for the 8th image of the sequence, because the corresponding file does not exist. Figure 3.6 shows a corresponding screenshot. The following code in the handler for the event `ImageGrabbed` reacts to this failure by stopping the application and then restarting the image sequence from the beginning:

```
Private Sub AVTView1_ImageGrabbed(hImage As HALCONXLib.HUntypedObjectX, _
                                  ByVal lError As Long)

  If hImage Is Nothing Then
    AVTView1.RunState = False
    Call AVTView1.InitGrabber(False)
  End If

End Sub
```

Let's return to the case the grabbing succeeded, i.e., ActivVisionTools sets the signal `'Processing Image'`. To assure hat the process does not "miss" this signal, a handshake is used. As no handshake is needed for the rest of the communication, it is enabled just before the the signal `'Processing Image'` is set, i.e., in the handler for the event `ProcessingStart` (see also figure 3.3 on page 33):

```
Private Sub AVTView1_ProcessingStart(atGlobalIconicData As _
                                     ActivVTools.AVTImageMessageClass)

  AVTDIOBoard1.EnableDIOHandshake = True

End Sub
```

The handshake is then disabled at the next suitable moment, i.e., in the handler for the event `Start` of `AVTDecision`:

```
Private Sub AVTDecision1_Start(atIconicData As _
                               ActivVTools.AVTImageMessageClass, _
                              cToolResults As Collection)

  AVTDIOBoard1.EnableDIOHandshake = False
End Sub
```

However, if a timeout occurs during the handshake, the execution cycle is aborted immediately, i.e., the event handler is not reached! In this case, ActivVisionTools would now expect a handshake for the signal `'Execution Cycle Aborted'`. To prevent the application from getting tangled in a sequence of handshake errors, handshaking is disabled in

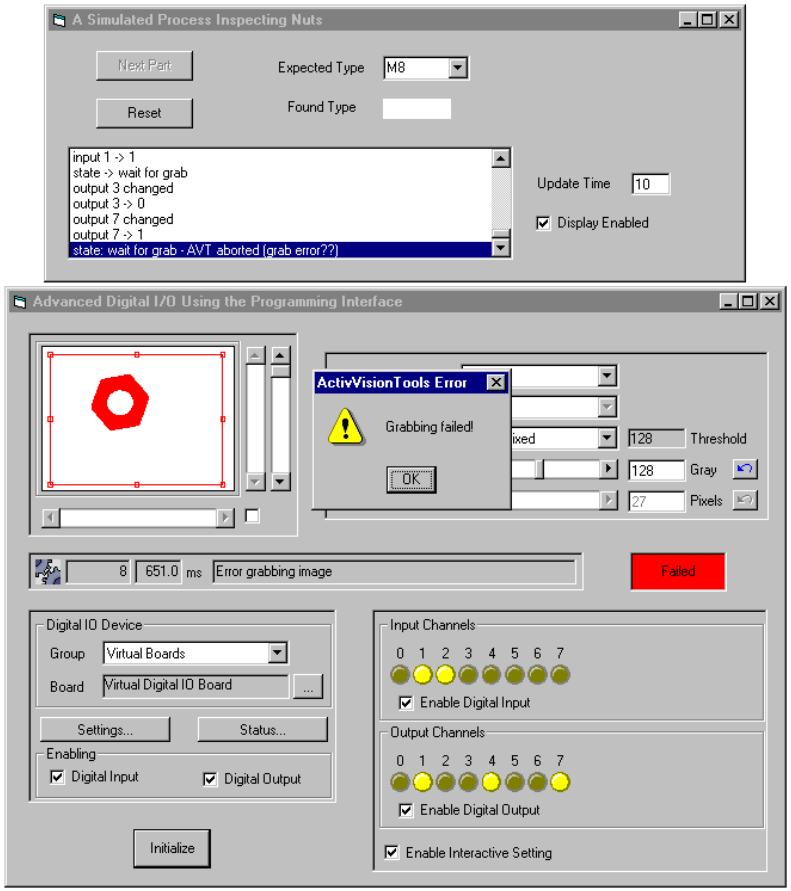



Figure 3.6: The effect of a grabbing error.

the handler for the event `ValueChanged` of the signal `'Execution Cycle Aborted'`; besides, the `RunState` is set to `False` to ensure that ActivVisionTools stops at the end of the execution cycle:

```
Private Sub atDIOAOAborted_ValueChanged(ByVal lNewValue As Long)

  If atDIOAOAborted.IsSignaled = True Then
    AVTDIOBoard1.EnableDIOHandshake = False
    AVTView1.RunState = False
  End If

End Sub
```

One problem remains in this case: As ActivVisionTools now definitely stops at the end of the execution cycle, the signal `'Execution Cycle Aborted'` is not cleared, because this is normally done at the start of a new cycle. In the example project, this problem is solved by allowing the process to clear the signal by setting the signal `'Stop Execution'`. This behavior is real;ized by adding the following code in the handler for the event `ValueChanged` of the signal `'Stop Execution'`:

```
Private Sub atDIOAOStopExec_ValueChanged(ByVal lNewValue As Long)

  If atDIOAOStopExec.IsSignaled = True Then
    Call atDIOAOAborted.SetSignal(False)
  End If

End Sub
```

To test the behavior in the case of a handshake timeout, open the dialog for output channel assignment and reduce the `Timeout` for the signal `'Processing Image'`.

4. **Specify expected type**

   After the handshake for the signal `'Processing Image'`, the process specifies the expected nut type using the corresponding two input signals, and then sets the signal `'DataValid'`. ActivVisionTools waits for this signal in the handler for the event `Start` of `AVTDecision`; if a timeout occurs, the excution cycle is aborted and a message is displayed in `AVTViewStatus`:

```
Private Sub AVTDecision1_Start(atIconicData As _
                               ActivVTools.AVTImageMessageClass, _
                              cToolResults As Collection)

  If atDIOAODataValid.IsSignaledWait = False Then
    AVTView1.AbortCycle = True
    Call AVTView1.SetStatusInfo("Timeout while waiting for DataValid!", _
                               eInfoStatus)
    Exit Sub
  End If
```

If no timeout occurred, the specified type is read

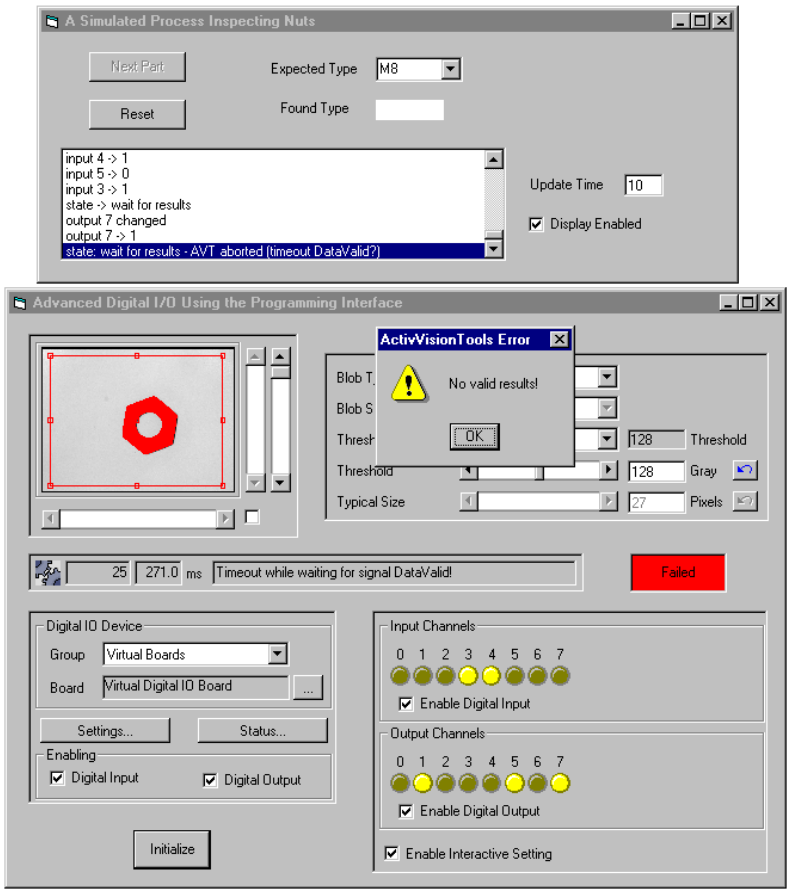



Figure 3.7: The effect of a timeout while waiting for `'DataValid'`.

```
If atDIOAOExpectedTypeLower.IsSignaled = True And _
   atDIOAOExpectedTypeUpper.IsSignaled = False Then
  bExpectNutM8 = True
ElseIf atDIOAOExpectedTypeLower.IsSignaled = False And _
   atDIOAOExpectedTypeUpper.IsSignaled = True Then
  bExpectNutM8 = False
End If
```

and the evaluation parameters are adapted accordingly:

```
If bExpectNutM8 = True Then
    Call atFeatureResult.SetEvaluator(atHandleMajorRadius, _
                                      dMinRadiusM8, dMaxRadiusM8, _
                                      eEvalIntNumber, eEvalOpInside, _
                                      False)
Else
    Call atFeatureResult.SetEvaluator(atHandleMajorRadius, _
                                      dMinRadiusM10, dMaxRadiusM10, _
                                      eEvalIntNumber, eEvalOpInside, _
                                      False)
End If
```

The data structure containing the results and the evaluation parameters for `AVTFeatureCalc` is accessed as follows (see the User's Manual for ActivFeatureCalc, section 4.2 on page 34, for more information about this data structure):

```
Dim atToolResult As AVTToolResult
Dim atFeatureResult As AVTToolResult
Dim atROIResult As AVTROIResult
Dim bFeatureResultsExist As Boolean


' wait for valid input signals describing expected type
If atDIOAODataValid.IsSignaledWait = False Then
  ' timeout! abort
  AVTView1.AbortCycle = True
  Call AVTView1.SetStatusInfo("Timeout while waiting for DataValid!", _
                              eInfoStatus)
```

To test the behavior in the case of a timeout, open the dialog for input channel assignment and reduce the `Timeout` for the signal `'DataValid'`. Figure 3.7 on page 40 shows a corresponding screenshot of the application.

5. **Output found type**
   After the image has been processed and the results evaluated, the signals describing the found type are set in the handler for the event `Start` of `AVTDIOBoard`:

```
Private Sub AVTDIOBoard1_Start(atIconicData As _
                               ActivVTools.AVTImageMessageClass, _
                              cToolResults As Collection)

  ' access results

  If atFeatureResult.Evaluation = True Then
    ' not necessary to output the type: output (0,0)
    Call atDIOAOIsTypeLower.SetSignal(False)
    Call atDIOAOIsTypeUpper.SetSignal(False)
  ElseIf atROIResult.ObjFeatureEvaluation(atHandleCompactness, _
                                          0) = False _
    Then
    ' unknown type: output (1,1)
    Call atDIOAOIsTypeLower.SetSignal(True)
    Call atDIOAOIsTypeUpper.SetSignal(True)
  Else
    If bExpectNutM8 = False Then
      ' not M8: output (1,0)
      Call atDIOAOIsTypeLower.SetSignal(True)
      Call atDIOAOIsTypeUpper.SetSignal(False)
    Else
      ' not M10: output (0,1)
      Call atDIOAOIsTypeLower.SetSignal(False)
      Call atDIOAOIsTypeUpper.SetSignal(True)
    End If
  End If

End Sub
```

The results of `AVTFeatureCalc` are accessed as described above.

## 3.2 How to Connect to Unsupported Digital I/O Boards

If you are using a digital I/O board that is not supported directly by ActivDigitalIO, you can nevertheless integrate it into ActivVisionTools. As described in section 1.3.1 on page 6, for the supported boards ActivVisionTools provides a special interface, which acts as a bridge between the SDK provided by the board manufacturer and the ActivVisionTools. In future versions, the programming interface of these digital I/O board interfaces will be open, so that you can integrate your board by creating a corresponding interface.

In the meantime, ActivVisionTools allows to integrate the access to digital I/O boards via a set of methods and events, which are described below. Figure 3.8 depicts the software architecture of this integration (compare with figure 1.3 on page 6).

As depicted in figure 3.8, the direct digital I/O interface to ActivVisionTools is kept very simple: `AVTDIOBoard` raises the event `OutputChanged`, whenever an output channel value changes; thus, you can attach code to actually perform the digital output in a corresponding event handler. On the other hand, you can inform ActivVisionTools about digital input via the method `DIOWriteInputBit` provided by the class `IAVTDIOBoardAccess`, which is an interface class of `AVTDIOBoard`.

The ActivVisionTools distribution includes the (completely preconfigured) Visual Basic project `unsupported\digital_unsupported.vbp`, which can serve as an example for integrating an unsupported digital I/O board. The digital I/O board is simulated by a Visual Basic user control. In this control, 8 buttons allow to set and clear input channels, while another set of 8 buttons is to display the state of the output channels, similarly to `AVTDIOBoardStatus` (see figure 3.9). Whenever an input changes, the control raises the event `InputChanged`; to set or clear output channels it provides the method `SetOutputChannel`.

This simulated digital I/O board is integrated into ActivVisionTools as follows: First, we need access to the interface of `AVTDIOBoard` mentioned above. For this purpose we declare a global variable

```
Private atDIOBA As IAVTDIOBoardAccess
```

and initialize it when the form is loaded:

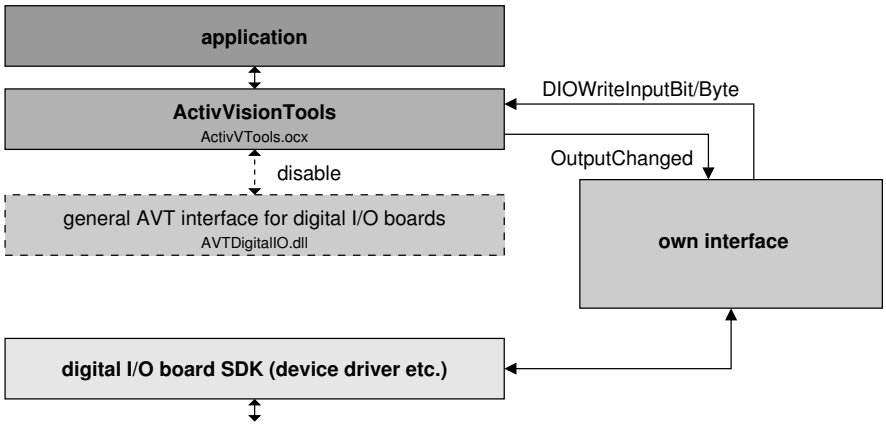



Figure 3.8: Integrating unsupported digital I/O boards.

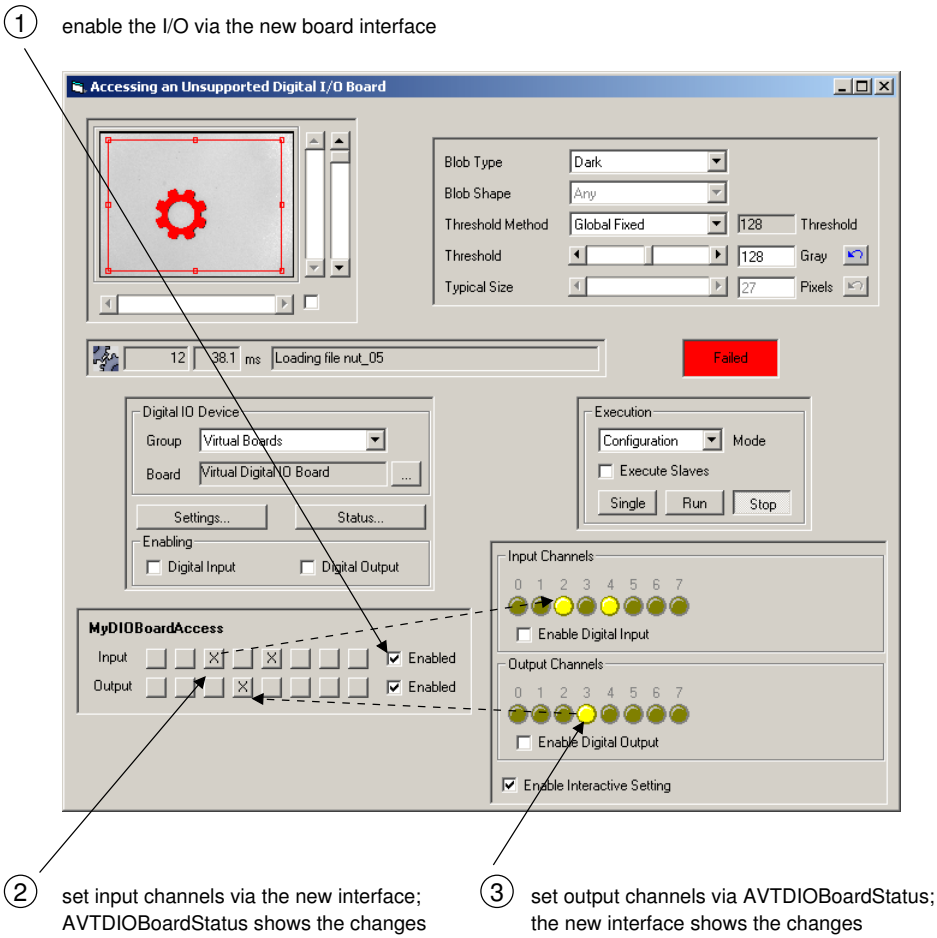



Figure 3.9: Using the virtual digital I/O board to create an own interface.

```
Private Sub Form_Load()

  Set atDIOBA = AVTDIOBoard1.Object

End Sub
```

To inform the simulated board about a changed output channel, we create a handler for the event `OutputChanged` of `AVTDIOBoard`. In it, we read the value of the output channel and “set” it via the `SetOutputChannel` of the control. Note that the event can also be raised once for multiple

changes; in this case, the parameter denoting the channel is set to -1:

```
Private Sub AVTDIOBoard1_OutputChanged(iPort As Integer, iChannel As Integer)

  Dim Channel As Integer, iValue As Byte

  If Not iChannel = -1 Then
    iValue = atDIOBA.DIOReadOutputBit(iChannel)
    Call MyDIOBoardAccess1.SetOutputChannel(iChannel, iValue)
  Else
    For iChannel = 0 To NumChannels - 1
      iValue = atDIOBA.DIOReadOutputBit(iChannel)
      Call MyDIOBoardAccess1.SetOutputChannel(iChannel, iValue)
    Next
  End If

End Sub
```

The task of digital input is solved even more easily by creating a handler for the event InputChanged of the control and then using the parameters in a call to the method DIOWriteInputBit of the interface to AVTDIOBoard:

```
Private Sub MyDIOBoardAccess1_InputChanged(iChannel As Integer, _
                                           iValue As Byte)

  Call atDIOBA.DIOWriteInputBit(iChannel, iValue)

End Sub
```

Please note that the low-level methods for setting and reading channels use the physical signal values (0 → "low", 1 → "high"), and not the logical values ('TRUE', 'FALSE')!

Finally, we optimize the performance of the direct digital I/O interface. The point is that even if you use the direct methods and events for digital I/O, ActivDigitalIO nevertheless creates and runs the digital I/O thread described in section 1.3.1 on page 6 and section 1.3.3 on page 11. To disable this thread, the following lines are inserted in the event handler Form_Load:

```
AVTDIOBoard1.EnableDIOInput = False
AVTDIOBoard1.EnableDIOOutput = False
```

Please note that disabling the digital input and output, be it via the methods as shown above or via the check boxes in AVTDIOBoard, only affects the digital I/O thread, not the ActivVisionTools thread.

Then, we enable the direct input with the following line:

```
atDIOBA.EnableInputDirect = True
```

Note that when using direct input, you should always disable the the digital I/O thread to prevent inconsistencies.